The Future Evolution of APEX & SWAT

Robin Taylor, Jaehak Jeong, Michael White & Jeff Arnold

Texas A&M University, Blackland Research Center
&
Grassland Soil & Water Laboratory,
Temple, Texas

Why Rebuild APEX & SWAT?

- APEX & SWAT have their origins in EPIC written in FORTRAN in the 1980s
- Newer versions of Fortran encourage more object-oriented code for easier maintenance & development of new modules
- Object-oriented code & other features facilitate data exchange between APEX & SWAT
- Harmonize APEX & SWAT databases
- Enable parallel processing

A Brief History of EPIC

Year	Feature Added
1981	Development of **EPIC** field model initiated at Temple, TX
1985	**EPIC** responds to 1985 Resources Conservation Act
1989	Expanded crop growth submodel
1991	Expanded root growth submodel, nitrogen fixation, pesticide fate
1992	CO_2 & vapor pressure effects on crops, multiple crops grown simultaneously
1993	**APEX** small watershed model & **SWAT** large watershed model for NPS
1994	Improved soil temperature submodel
1995	Improved weather generator, water table dynamics submodel, water runoff, nitrification-volatilization
1997	RUSLE water erosion equation, snowmelt runoff & erosion
1998	WESS wind erosion model, Baier-Robertson evapotranspiration
2000	Green & Ampt infiltration function
2004	Century model of carbon & potassium cycling
2010+	Spatial weather generator, Southern Oscillation, pest population dynamics

A Brief History of Fortran

Year	Name	Feature Added
1953	FORTAN	*The IBM Mathematical **For**mula **Tran**slating System* released.
1958	FORTRAN II & III	Procedural programming (CALL, SUBROUTINE, FUNCTION, RETURN)
1961	FORTRAN IV	Boolean expressions (.AND., .OR., .EQ., .NE., etc.)
1966	FORTRAN 66	Labeled & unlabeled COMMON & EQUIVALENCE
1977	FORTRAN 77	Block structures (IF, THEN, ELSE)
1991	Fortran 90	Recursion, Pointers, Dynamic memory (ALLOCATE), Operator overloading (INTERFACE), Derived data types (TYPE), Structured multi way selection (SELECT CASE)
1995	Fortran 95	Incremental revision to Fortran 90
2003	Fortran 2003	Inheritance, Procedure pointers
2008	Fortran 2008	Enhanced parallel processing features

Fortran 1991 Features

- **Dynamic Memory Allocation**
  - Runtime creation of arrays
- **Pointers**
  - Manipulate addresses for efficient memory access
- **Derived Data Types**
  - Storage and organization of a variety of different data types in a single efficient record
- **Operator Overloading**
  - Associate mathematical operations with a keyword
- **Recursion**
  - Enable a function to call itself

Powerful Combinations of Features

- **Derived Types + Pointers** = Linked Lists (Relational Databases & Nested Data Structures)
- **Derived Types + Overloading** = Arithmetic on Relational Databases with special-purpose Operators
- **Dynamic Memory + Recursion** = Arithmetic on Nested or Hierarchical Data Structures

Application to EPIC/APEX/SWAT

- Dynamic Memory Allocation & Pointers
  - Runtime creation of arrays:

```
REAL*4, ALLOCATABLE :: SALB(:),HSG(:),FFC(:),&
   .
  Z(:,:),SAN(:,:),SIL(:,:),BD(:,:),UW(:,:),FC(:,:)&
   .
  WHSN(:,:),WHPN(:,:)
   .
   .
NST = 100                ! Number of Soil Types
NSL = 15                 ! Number of Soil Layers
   .
   .
ALLOCATE (SALB(NST),HSG(NST),FFC(NST),STAT=Ierr)
ALLOCATE (Z(NST,NSL),SAN(NST,NSL),SIL(NST,NSL),STAT=Ierr)
```

Application to EPIC/APEX/SWAT

➢ Derived Data Types

A vector for every property

```
INTEGER, PARAMETER :: $NST = 100      ! Soil Types
INTEGER, PARAMETER :: $NSL = 15       ! Soil Layers

COMMON /SOILS/ SALB($NST),HSG(&NST),FFC($NST),&
  WTMN($NST),WTMX($NST),WTBL($NST),GWST($NST),&
  GWMX($NST),RFTT($NST),RFPK($NST),TSLA($NST),&
  XIDS($NST),RTN1($NST),XIDK($NST),ZQT($NST),&
  ZF($NST),ZTK($NST),FBM($NST),FHP($NST),&
  Z($NST,$NSL),BD($NST,$NSL),UW($NST,$NSL),&
  FC($NST,$NSL),SAN($NST,$NSL),SIL($NST,$NSL),&
  WON($NST,$NSL),PH($NST,$NSL),SMB($NST,$NSL),&
  WOC($NST,$NSL),CAC($NST,$NSL),CEC($NST,$NSL),&
  ROK($NST,$NSL),CNDS($NST,$NSL),SSF($NST,$NSL),&
  RSD($NST,$NSL),BDD($NST,$NSL),PSP($NST,$NSL),&
  SATC($NST,$NSL),HCL($NST,$NSL),WPO($NST,$NSL),&
  EXCK($NST,$NSL),ECND($NST,$NSL),STFR($NST,$NSL),&
  ST($NST,$NSL),CPRV($NST,$NSL),CPRH($NST,$NSL),&
  WLS($NST,$NSL),WLM($NST,$NSL),WLSL($NST,$NSL),&
  WLSC($NST,$NSL),WLMC($NST,$NSL),WLSLC($NST,$NSL),&
  WLSLNC($NST,$NSL),WBMC($NST,$NSL),WHSC($NST,$NSL),&
  WHPC($NST,$NSL),WLSN($NST,$NSL),WLMN($NST,$NSL),&
  WBMN($NST,$NSL),WHSN($NST,$NSL),WHPN($NST,$NSL)
```

Properties of a soil structure

```
INTEGER, PARAMETER :: $NSL = 15

TYPE LAYERS
  REAL*4 Z
  REAL*4 SAN
  REAL*4 SIL
     .
     .
  REAL*4 WHPN
END TYPE LAYERS

TYPE SOILS
  REAL*4 SALB
  REAL*4 HSG
  REAL*4 FFC
     .
     .
  REAL*4 FHP
  TYPE(LAYERS) Layer($NSL)
END TYPE SOILS

TYPE(SOILS), POINTER :: Soil(:)
COMMON /SOILS/ Soil
```

Application to EPIC/APEX/SWAT

- Derived Types + Pointers .... Linked List declaration

```
TYPE SOILS
  INTEGER      ID
  CHARACTER*32 Name
  UNION
    MAP
      REAL*4   Global(19)
    END MAP
    MAP
      REAL*4   SALB         ! Soil albedo
      REAL*4   HSG          ! Hydrologic soil group code (1=A; 2=B; 3=C; 4=D)
          .
          .
      REAL*4   FHP          ! Fraction of humus in passive pool (0.3-0.7)
    END MAP
  END UNION
  INTEGER      NSL          ! Number of Soil Layers
  TYPE(LAYERS), POINTER :: Layer(:)  ! Pointer to soil layer structure
  TYPE(SOILS),  POINTER :: Next      ! Pointer to next soil type in list
END TYPE SOILS
TYPE(SOILS), POINTER :: Soil(:),ThisSoil
COMMON /SOILS/ Soil,ThisSoil
```

Application to EPIC/APEX/SWAT

- Derived Types + Pointers .... Reading data

```
         .
         .
       ALLOCATE(Soil)                                   ! For first soil
       ThisSoil => Soil
       NULLIFY(ThisSoil%Next)
       DO WHILE(.TRUE.)
         READ(K,*,END=10,ERR=999)ThisSoil%ID,ThisSoil%NSL
         IF (ThisSoil%ID.EQ.0) GOTO 10
         READ(K,'(A32)',END=10,ERR=999)ThisSoil%Name
         READ(K,*,END=10,ERR=999)ThisSoil%Global
         ALLOCATE(ThisSoil%Layer(ThisSoil%NSL))         ! Allocate Soil Layers
         READ(K,*,END=999,ERR=999)ThisSoil%Layer
         ALLOCATE(ThisSoil%Next)                        ! For each subsequent soil
         ThisSoil => ThisSoil%Next
         NULLIFY(ThisSoil%Next)
       ENDDO
10     CONTINUE
       ThisSoil => Soil
         .
         .
999    ! Output Error condition
```

Application to EPIC/APEX/SWAT

- Derived Types + Pointers .... Independent vectors

```
      CALL NPMIN(ISL,ISA)              ! Operate on Layer ISL of soil in Subarea ISA
        .
        .
      SUBROUTINE NPMIN(ISL,ISA)
!     APEX0806
!     NPMIN computes phosphorus flux between the soluble, active mineral
!           & stable mineral P pools.
      USE PARM            ! BK,PSP,WPMA,WPML,WPMS defined in MODULE PARM
!
      RTO = MIN(0.8,PSP(ISL,ISA)/(1.0 - PSP(ISL,ISA)))
      RMN = PRMT(84)*(WPML(ISL,ISA) - WPMA(ISL,ISA)*RTO)
      X1 = 4.0*WPMA(ISL,ISA) - WPMS(ISL,ISA)
      IF( X1.GT.500.) THEN
          ROC = 10.0**(LOG10(BK(ISL,ISA)) + LOG10(X1))
      ELSE
          ROC = BK(ISL,ISA)*X1
      ENDIF
      ROC = PRMT(85)*ROC
      WPMS(ISL,ISA) = WPMS(ISL,ISA) + ROC
      WPMA(ISL,ISA) = WPMA(ISL,ISA) - ROC + RMN
      WPML(ISL,ISA) = WPML(ISL,ISA) - RMN
!
      RETURN
      END
```

Application to EPIC/APEX/SWAT

- Derived Types + Pointers .... Using data structures

```
      CALL NPMIN(ThisSoil,L)        ! Operate on Layer L of the current soil
        .
        .
      SUBROUTINE NPMIN(Soil,L)
!     APEX0806
!     NPMIN computes phosphorus flux between the soluble, active mineral
!           & stable mineral P pools.
      INCLUDE ‘Structures.fd’ ! TYPE(SOILS) defined in Structures.fd
      TYPE(SOILS) Soil        ! Local variable aliased with the dummy argument
      INTEGER L               ! The current Layer
!
      RTO = MIN(0.8,Soil%psp(L)/(1.0 - Soil%psp(L)))
      RMN = PRMT(84)*(Soil%wpml(L) - Soil%wpma(L)*RTO)
      X1 = 4.0* Soil%wpma(L) - Soil%wpms(L)
      IF (X1.GT.500.) THEN
          ROC = 10.0**(LOG10(Soil%BK(L)) + LOG10(X1))
      ELSE
          ROC = Soil%bk(L)*X1
      ENDIF
      ROC = PRMT(85)*ROC
      Soil%wpms(L) = Soil%wpms(L) + ROC
      Soil%wpma(L) = Soil%wpma(L) - ROC + RMN
      Soil%wpml(L) = Soil%wpml(L) - RMN
!
      RETURN
      END
```

Application to EPIC/APEX/SWAT

- Derived Types + Operator & Function Overloading
  - Create operators such as .Mean. and .Variance.
  - Useful for defining output routines
    - Daily variables
    - Monthly averages
    - Annual averages

Example:

```
Month(M)%SoilP(L) = Subarea(K)%Soil%P(L).Mean.Mnth
Annual%SoilP(L)   = Subarea(K)%Soil%P(L).Mean.Year
```

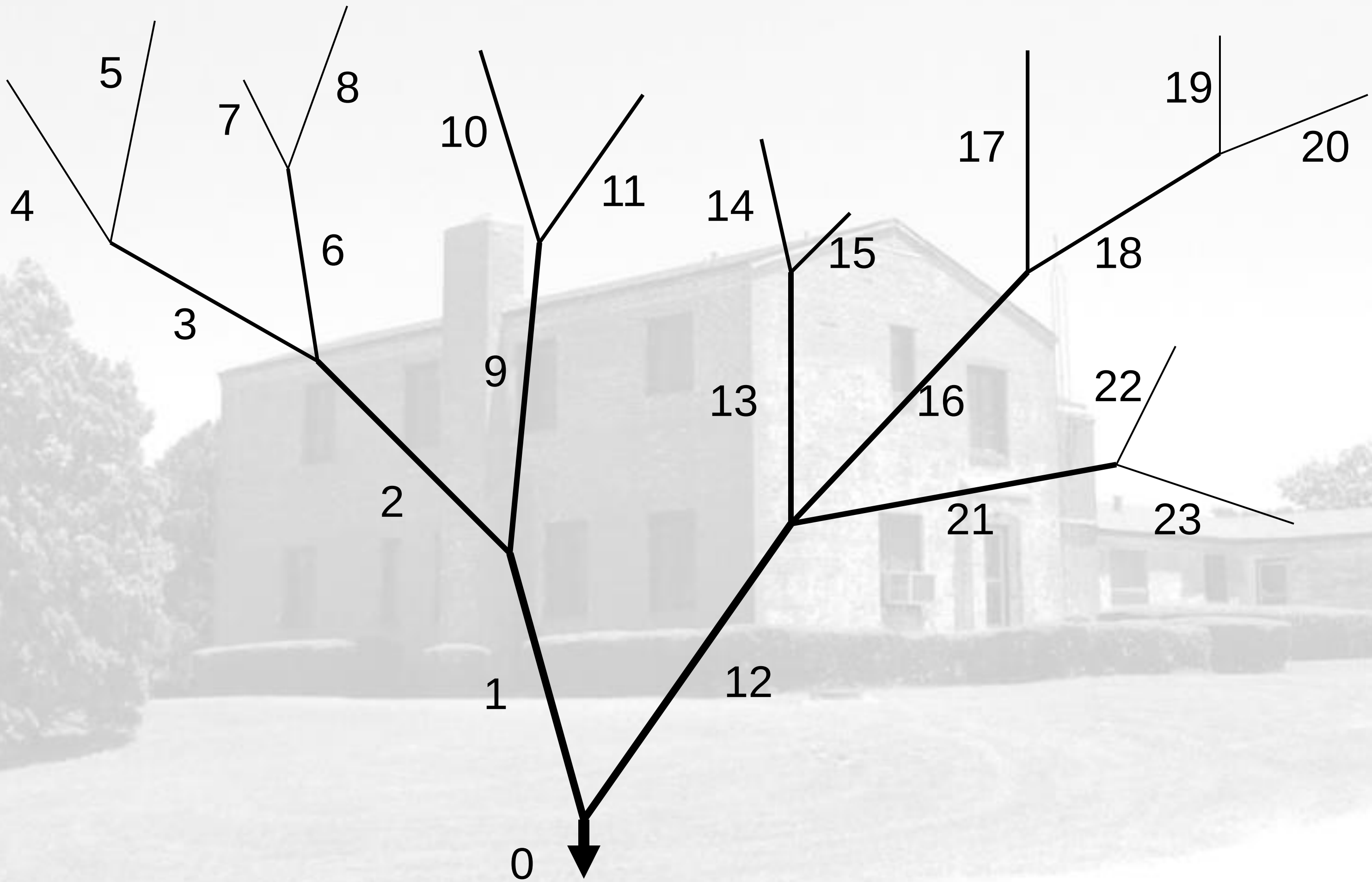
Application to EPIC/APEX/SWAT

- Derived Types + Pointers + Recursion = Operations on Nested (Hierarchical) Objects

```
      TYPE SUBAREA
        INTEGER                 IDSA                  ! Subarea identifier
        CHARACTER*40            Name                  ! Subarea name
        TYPE(SOILS)             Soil                  ! Soil characteristics
        TYPE(WATER)             Water                 ! Water balance
        TYPE(NUTRS)             Nuts                  ! Nutrients (sediment, C, N, P & K)
        INTEGER                 NCrops                ! Number of crops
        TYPE(CROPS), POINTER :: Crop(:)               ! Status of Crop(s)
        TYPE(SCHED), POINTER :: Sched(:)              ! Management schedule
!                                                     ! Cross-referenced with Crops
        TYPE(SUBAREA), POINTER :: Outlet              ! Pointer to receiving subarea
                                                      !  (NULL if watershed outlet)
        TYPE(SUBAREA), POINTER :: Inlet(:)            ! Pointer to upstream subarea(s)
                                                      !  (NULL if headwater subarea)
        TYPE(FLOW)              Inflow                ! Inputs from upstream subarea(s)
        TYPE(FLOW)              Outflow               ! Outflow to downstream subarea
      END TYPE SUBAREA
!
      TYPE(SUBAREA), POINTER :: Subs(:),ThisSub
      COMMON /SUBAREA/ Subs,ThisSub
        .
        .
      ALLOCATE Subs(Site%NSA)                 ! Number of Subareas is a Site property
```

Application to EPIC/APEX/SWAT

- Order of execution in a recursive watershed



Application to EPIC/APEX/SWAT

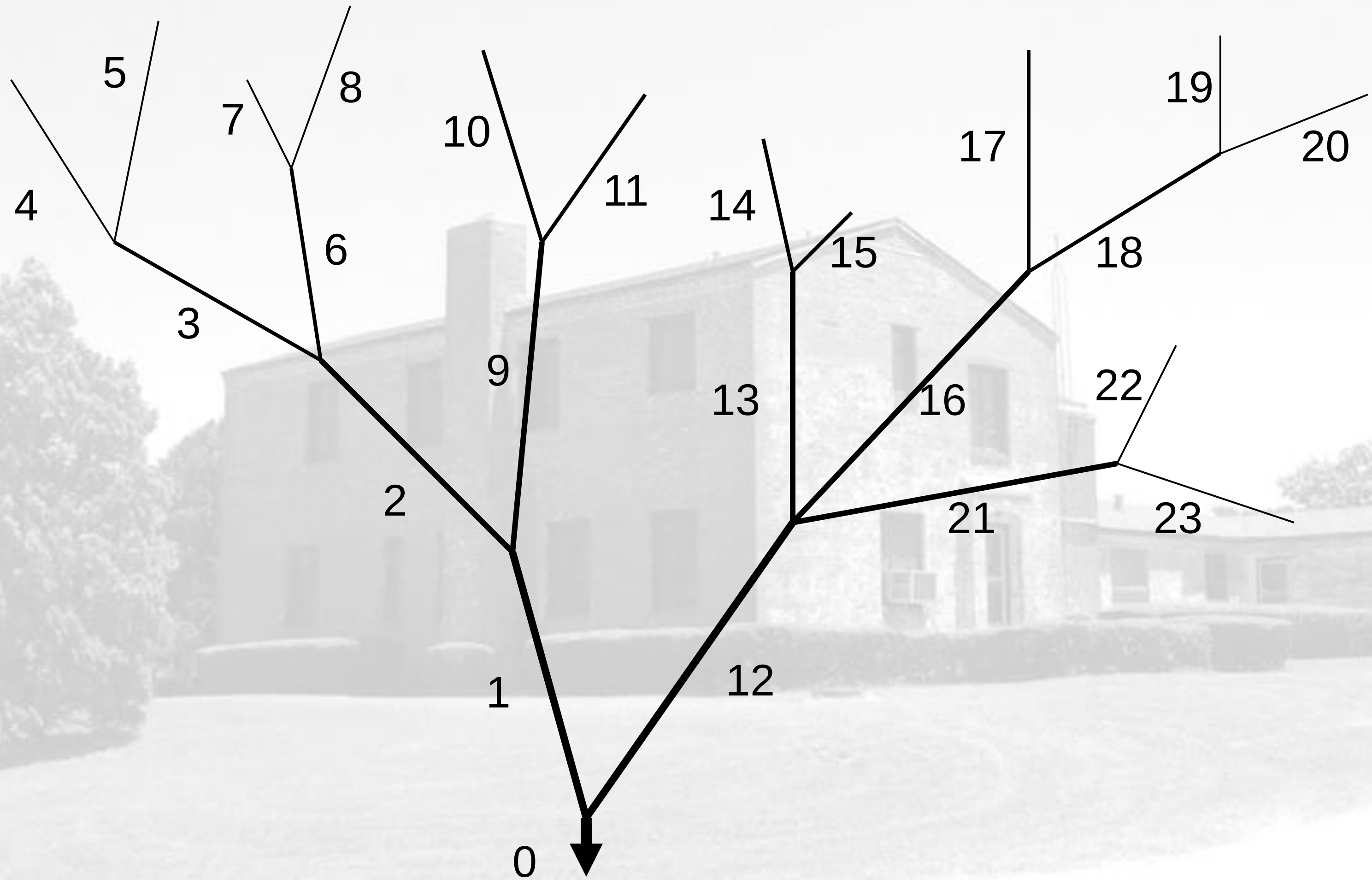
- Derived Types + Pointers + Recursion = Operations on Nested (Hierarchical) Objects

```
      RECURSIVE SUBAREA InitializeSA(Subarea)
      TYPE(SUBAREA) Subarea
      TYPE(SUBAREA), POINTER :: &
          Upstream,Downstream
!
      Upstream   => Subarea%Inlet
      Downstream => Subarea%Outlet
!
!     Initialize accumulators
      Downstream%Inflow = 0
!
      DO WHILE (ASSOCIATED(Upstream)
        CALL InitializeSA(Upstream)
      ENDDO
!
      RETURN
      END
```

```
      RECURSIVE SUBROUTINE ProcessSA(Subarea)
      TYPE(SUBAREA) Subarea
      TYPE(SUBAREA), POINTER :: &
          Upstream,Downstream
!
      Upstream   => Subarea%Inlet
      Downstream => Subarea%Outlet
      Subarea%Outflow = 0
!
!     CALL SUBROUTINE here to compute subarea
!         functions & save in Subarea%Outflow
!
!     Now save this subarea's contribution
!     downstream:
      Downstream%Inflow = Downstream%Inflow &
                        + Subarea%Outflow
!
      DO WHILE (ASSOCIATED(Upstream)
        CALL ProcessSA(Upstream)
      ENDDO
!
      RETURN
      END
```

Application to EPIC/APEX/SWAT

- Order of execution in a recursive watershed



Status of EPIC/APEX/SWAT

- SWAT
  - Phase 1: Modular construction – done
  - Phase 2: Derived Data Types – underway
- EPIC
  - Phase 1: Derived Data Types – done
  - Phase 2: Modular construction – underway
- APEX
  - As for EPIC
  - Recursive construction – planned

Some of the people working on the Future Evolution of APEX & SWAT

Questions?